A VZ-Epson printer patch — the search continues

Larry Taylor

Part 2

IN THE PREVIOUS instalment, printing of the VZ's inverse and graphics characters had been made possible. At this point, the ideal enhancement to our printer patch would be to enable the VZ's COPY command to function correctly when matched with an EPSON type printer. This should be possible, but we must first examine why the usual means for intercepting BASIC key words, during programme execution, won't work in the case of the COPY command.

The VZ's ROM owes much to that used in the earlier TRS-80 computers. The COPY routine, however, is one of a number of additions which greatly enhance the VZ's capabilities. As such, it contains none of the DOS exits, which are to be found in the older sections of the ROM. These exits, or "vectors", are calls to an area in the communications area of RAM, and provide the means by which some BASIC commands may be altered or redirected. Since the VZ DOS makes no use of these vectors, none have been provided in the newer sections of the ROM. My initial hopes dashed, I began to investigate the method used to integrate the DOS into the VZ's operating system. In doing so, I uncovered an alternative vector, one which would make it possible for us to not only intercept the COPY command, but also open the door to further enhancements to the VZ's BASIC.

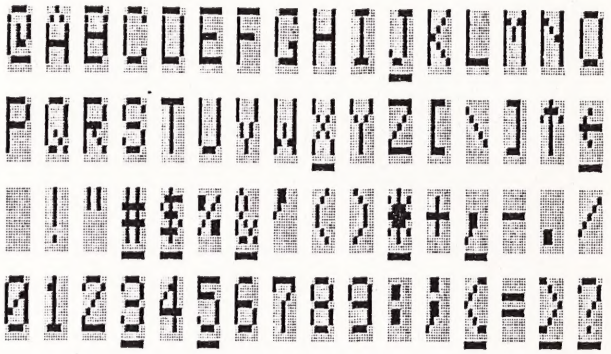
How so?

It is important to understand, initially, why this type of modification is possible. When we write a BASIC programme, we are creating what we hope will be a precise set of instructions. Unfortunately, before the computer can understand and respond to our commands, each instruction in turn has to be painstakingly translated or intrpreted. This is the reason for BASIC's slowness, and it can really only be effectively overcome by having the programme translated or compiled prior to execution. Yet, because a BASIC programme is interpreted as it runs, it is possible that additional commands may be added to the language, provided they are intercepted and executed prior to reaching the VZ's own interpreter. This is precisely what happens when a disk operating system is added. New commands enabling disk operations to be performed, supplementing the existing BASIC. In the case of the COPY command, we are seeking to redirect it to a routine compatible with EPSON type printers, and on completion, have it return as though all had proceeded normally.

As I undertook to produce this extension to the patch, I found myself venturing much further than I had originally intended. The project involved modifying the existing ROM routine, as well as enhancing the COPY command to provide for a second screen dump routine of my own design. Furthermore, I allowed for a copy of the LO-RES screen without the usual linefeeds. I also sought to eliminate those unfortunate flaws in the inverse character data. Listing 1, which was kindly supplied by Bob Kitch, enables a closer examination of the inverse characters held in ROM, by displaying them on the HIRES screen. By relocating the ROM table to RAM at the top of memory the necessary modifications to the data have been made possible.

VZ ROM, INVERSE CHARACTER SHAPE TABLE

VZ ROM, PRINTER PATCH MODIFIED TABLE



(note changes to underlined characters)

The accompanying illustration allows a comparison to be made between the ROM characters, at top, and those in the shape table addressed by the printer patch. Incidentally, should you decide that you still don't like the look of the amended characters, it is possible, using the same approach, to either further refine them, or even custom design a completely new set.

Inspired at having overcome this obstacle, and because I have written a number of programs using an Extended BASIC, I wanted the routine to be able to list those commands, which would not normally be recognised. The final aim was to deal with the printer's unimpressive performance, signalled by a dramatic decrease in speed, each time it had to print a graphics or inverse character. The solution I chose to minimise these delays was to feed the data into a section of RAM, which would act as a collection area or buffer, prior to printing. A discussion in detail of how each of these refinements was implemented would only serve to complicate what is otherwise a relatively straightforward procedure. I have elected, instead, to demonstrate how to intercept and enhance an existing keyword on a smaller scale by using another of the VZ's commands.

1 of 3.

Enhanced CLS

Tandy's Colour Computer has an enhanced CLS command which enables the user to clear the screen to any one of nine background colours. The syntax is CLSn, where n may be a number in the range 0-8. To illustrate how enhancements to the existing language can be accomplished, this command will be necessary to examine further how the VZ operates.

When a BASIC program is RUN, control passes to a machine language ROM routine, the Execution Driver at 1D5AH, which scans each line of the BASIC programme as it comes to it and begins to translate it. Part of the translation process involves looking for tokens. These are values in the range 128-250 (80H-FAH) that take the place of BASIC reserved words e.g: CLS = 132 (84H). Once the word has been identified and checked for correct syntax, control is passed to the corresponding ROM routine before returning to continue the translation.

On power-up, the address of the routine which examines each byte in a line of BASIC, is stored at 7804H. This is the vector hinted at earlier, and in a non-disk VZ it will normally contain a pointer to the RST 10H routine at 1D78H. Because this vector is in RAM it can be easily changed. This was done so that at a later stage the DOS could be included.

At least three different versions of the VZ DOS could be included that I am aware of, and two of these display the same version number on power up. Consequently, the only fixed location common to all three versions is a jump table commencing at 4005H. This makes it difficult to refer to an actual address within the DOS, where command processing is carried out. However, since all processing must be channelled via the above-mentioned vector, a peek at this address will uncover the whereabouts of the DOS interpreter. A close examination of this region of the DOS will reveal how the added disk commands are interpreted and implemented. This information will enable us to introduce into the system an enhanced command of our own choosing. The trick is to ensure that, as far a the VZ's interpreter is concerned, nothing unusual has happened.

The accompanying assembly language programme in Listing 2, with its associated comments, shows in greater detail how this is accomplished. If you do not have access to an Editor Assembler, Listing 3 is a BASIC version, which pokes the routine into memory. Having adjusted the top of memory pointer, the address at 7804H is stored and replaced by our own. The programme then locates the new routine at the top of the memory. Now each time a byte is to be examined during execution it must first pass through our checkpoint. Once the origin of the call is established, the routine looks for the CLS token, 132 (84H).

Only when it has been located does the routine proceed to examine the next byte. This is checked to see if it lies in the range 0-9. Once it has passed this test, the clear screen routine is implemented, after first calculating the appropriate value, with which to fill the screen. You will notice that not only is it necessary to check for the new command, but also to provide the routine which implements it. In this case a simple block load to the screen has been used. Control is then returned to the ROM processing routine, which prepares to examine the byte following ournew command. So, as far as the VZ knows, everything is continuing normally. Tricky isn't it?

The VZ will now respond to the CLSn command, when entered, either directly from the keyboard, or from within a program, with one exception. For some unexplained reason, during IF-THEN-ELSE processing the ROM accesses the byte examine routine at 1D78H directly, instead of via a RST 10H call. This means there is no efficient method for our programme to intercept the new command, when it is used in an IF-THEN-ELSE statement. The problem can best be overcome, by means of a minor change in syntax, when entering the programme line. Using the line,

100 IF X=4 THEN CLS4

should clear the screen to red, when X=4.

What actually happens is that the screen clears normally, followed by a SYNTAX ERROR message, indicating the routine at 1D78H has not recognised our enhanced command. ▶

LISTING 1

```
10  '          ************************************
20  '          ***   DISPLAY INVERSE CHARACTER   ***
30  '          ***          SET IN ROM           ***
40  '          ***     AS USED BY DOT MATRIX     ***
50  '          ***          PRINTER              ***
60  '          ***     R. B. KITCH   27/1/86     ***
70  '          ************************************
80  '
100 'WHEN INVERSE CHARACTERS ARE SENT TO A DOT MATRIX PRINTER
110 'THE PRINTER SHIFTS TO GRAPHICS MODE AND REQUIRES A ROUTINE
120 'TO SUPPLY THE APPROPRIATE SHAPES TO THE HEAD. (NORMAL
130 'CHARACTERS ARE HELD IN THE PRINTERS ROM)
140 'IN THE VZ COMPUTER A TABLE OF SHAPES IS LOCATED AT
150 '3B94H TO 3CD3 IN ROM. THERE ARE 64 CHARACTERS, EACH USING
160 '5 BYTES TO DEFINE THEIR GRAPHIC SHAPE. THE SHAPES MAY BE
170 'DECODED AND OUTPUT TO THE SCREEN AS IS DONE IN THIS
180 'PROGRAM. NOTE THAT THERE ARE SOME ERRORS IN THE ROM.
190 'THE 5 BYTES DEFINE A 5 BY 8 DOT MATRIX WHICH IS THE SHAPE
200 'OF THE CHARACTER, WHICH INCIDENTALLY ARE NOT ORDERED
210 'ACCORDING TO THE ASCII CODE.
220 'THE FIRST BYTE DEFINES THE LEFT HAND EDGE OF THE CHARACTER-
230 'WHICH IS THE FIRST PRINTED DURING A PASS OF THE PRINTER
240 'HEAD. IN TANDY PRINTERS THE MSB IS THE LOWERMOST PIN OF THE
250 'HEAD AND THE LSB IS THE UPPERMOST PIN. THE PINS ON EPSON
260 'PRINTER HEADS ARE ARRANGED IN THE OPPOSITE SENSE. THIS
270 'REQUIRES THAT THE BITS IN EACH BYTE BE REVERSED.
280 '************************************************************
290 '
300 DIM MK%(7) :'***VECTOR OF BIT MASK VALUES - POWERS OF 2
310 DIM BT%(7) :'***VECTOR OF DECODED BITS FROM ROM VALUE.
320 '
330 '***FILL MASK VECTOR WITH POWERS OF 2 FOR DECODING.
340 FOR I%=0 TO 7 :MK%(I%)=2^I% :NEXT I%
350 '
400 '***INITIALIZE PARAMETERS - MAY BE CHANGED TO VARY SCREEN.
410 CC%=4     :'***CHARACTER COLOUR.  (1-4)
420 BC%=2     :'***BACKGROUND COLOUR. (1-4)
430 CS%=0     :'***COLOUR SET. (0-1)
440 CW%=3     :'***COLUMN WIDTH BETWEEN CHARACTERS.
450 SP%=16    :'***ROW SPACING FOR CHARACTERS.
460 HS%=0     :'***STARTING HORIZONTAL POSITION ON HI-RES SCREEN.
470 VP%=3     :'***STARTING VERTICAL POSITION ON HI-RES SCREEN.
480 HM%=127   :'***MAXIMUM HORIZONTAL POSITION. (0-127)
490 '
600 '***SET UP MAIN LOOP TO STEP THROUGH ROM FROM 3B94H-3CD3.
610 BK%=0                       :'***BYTE COUNTER FOR EACH CHARACTER.
620 HP%=HS%                     :'***SET HORIZONTAL POSITION TO START
630 MODE(1) :COLOR,CS%          :'***SET HI-RES SCREEN AND COLOR SET.
640 SM%=15252                   :'***START OF SHAPE TABLE
650 EM%=15571                   :'***END OF SHAPE TABLE
660 FOR AD%=SM% TO EM%          :'***ADDRESSES FOR SHAPE TABLE.
670   DV%=PEEK(AD%)             :'***DECIMAL VALUE READ FROM TABLE
680 '
700 '***DECODE THE INDIVIDUAL BITS OF DV% AND STORE IN BT%().
710 '***THE MASK VALUES IN MK%() ARE "ANDED" WITH THE VALUE.
720 '***THE RESULT STORED IN BT%() IS THE "COLOUR" OF THE BIT.
730   FOR I%= 0 TO 7            :'***PROCEED FROM LSB TO MSB.
740     IF DV% AND MK%(I%) THEN BT%(I%)=BC% ELSE BT%(I%)=CC%
750   NEXT I%
800 '
810 '***CHECK THAT THERE IS ENOUGH ROOM TO PLOT CHARACTER.
820   IF BK%=0 AND HM%-HP%<4 THEN HP%=HS% :VP%=VP%+SP%: '*NEW ROW
830   BK%=BK%+1                 :'***INCREMENT BYTE COUNTER.
840 '
900 '***OUTPUT BYTE TO SCREEN.
910   FOR I%=0 TO 7
920     COLOR BT%(I%)           :'***SET COLOUR OF BIT.
930     SET(HP%,VP%+I%)         :'***PLOT BIT.
940   NEXT I%
950 '
1000 '***PREPARE FOR NEXT BYTE.
1010   HP%=HP%+1                :'***INCREMENT HORIZONTAL POSITION.
1020   IF BK%=5 THEN BK%=0 :HP%=HP%+CW%      :'***NEW CHARACTER.
1030 NEXT AD%
2000 GOTO 2000 :END
```

LISTING 1A

```
100 'THIS SHORT LISTING CAN BE USED BY OWNERS OF THE PRINTER
110 'PATCH TO CALCULATE THE START AND END LOCATIONS OF THE
120 'REVISED INVERSE CHARACTER SHAPE TABLE IN THE COMPLETED
130 'VERSION. BY SUBSTITUTING THE NEW VALUES FOR THOSE WHICH
140 'APPEAR IN LINES 640 AND 650 OF LISTING 1, THE MODIFIED
150 'CHARACTERS CAN BE DISPLAYED ON THE HIRES SCREEN.
160 '************************************************************
170 '
180 '***CALCULATE THE TOP OF MEMORY
190 TM=PEEK(30897)+256*PEEK(30898)
200 IF TM>32767 THEN TM=TM-65536
210 '
220 '***ADD OFFSET TO TOP OF MEMORY TO LOCATE START OF TABLE
230 SM%=TM+666     :'***START OF SHAPE TABLE.
240 '
250 '***ADD 64 CHARACTERS X 5 BYTES TO LOCATE END OF TABLE
260 EM%=SM%+64*5-1 :'***END OF SHAPE TABLE
270 '
280 '***PRINT START AND END ADDRESSES
290 PRINT"START - SM%=";SM%
300 PRINT"END   - EM%=";EM%
```

LISTING 2

```
0001 ;************************
0002 ;* ENHANCED CLS COMMAND *
0003 ;* BY LARRY TAYLOR 1986 *
0004 ;************************
0005 ;
0006 ;THIS SECTION RELOCATES
0007 ;THE PROGRAM TO THE TOP
0008 ;OF AVAILABLE MEMORY.
0009 ;
0010 VCTR EQU  7A28H          ;SET VCTR AS 7A28H
0011      LD   SP,7700H       ;LOAD STACK POINTER
0012      LD   HL,(7881H)     ;GET THE TOP OF MEMORY
0013      LD   BC,ENDP-NVCT   ;GET LENGTH OF PROGRAM
0014      PUSH BC             ;SAVE PROGRAM LENGTH
0015      XOR  A              ;RESET ALL FLAGS
0016      SBC  HL,BC          ;TAKE LENGTH FROM TOP OF MEMORY
0017      LD   (7881H),HL     ;LOAD NEW TOP OF MEMORY
0018      PUSH HL             ;SAVE NEW TOP OF MEMORY
0019      XOR  A              ;RESET ALL FLAGS
0020      LD   BC,33H         ;RESERVE 50 BYTES STRING SPACE
0021      SBC  HL,BC          ;TAKE SPACE FROM TOP OF MEMORY
0022      LD   (78A0H),HL     ;LOAD START OF STRING SPACE
0023      POP  DE             ;RETRIEVE TOP OF MEMORY
0024      INC  DE             ;INCREASE BY ONE
0025      LD   HL,(7804H)     ;GET CURRENT RST10H VECTOR
0026      LD   (VCTR),HL      ;STORE IT IN 7A28H
0027      LD   (7804H),DE     ;LOAD NEW VECTOR
0028      LD   HL,NVCT        ;GET START OF PROGRAM TO MOVE
0029      POP  BC             ;RETRIEVE PROGRAM LENGTH
0030      LDIR                ;MOVE TO NEW LOCATION
0031      CALL 1B4DH          ;DO A NEW
0032      JP   1A19H          ;JUMP TO READY MESSAGE
0033 ;
0034 ;START OF THE PROCESSING
0035 ;ROUTINE FOR NEW COMMAND.
0036 ;
0037 NVCT EXX                ;SAVE ALL REGISTERS
0038      LD   HL,1D5BH       ;CHECK TO
0039      POP  DE             ;SEE IF THE
0040      OR   A              ;RETURN
0041      SBC  HL,DE          ;ADDRESS
0042      PUSH DE             ;IS 1D5BH
0043      EXX                 ;RESTORE ALL REGISTERS
0044      JP   NZ,1D78H       ;IF NOT GO TO NORMAL PROCESSING
0045      PUSH HL             ;SAVE STRING ADDRESS
0046      CALL 1D78H          ;GET NEXT VALUE FROM STRING
0047      JR   NZ,CONT        ;IF NOT ZERO THEN CONTINUE
0048 POP  POP  HL             ;ELSE RESTORE STRING ADDRESS
0049      LD   DE,(VCTR)      ;RETRIEVE ORIGINAL VECTOR
0050      PUSH DE             ;AND JUMP
0051      RET                 ;TO IT
0052 CONT CP   84H            ;CHECK FOR CLS TOKEN
0053      JR   NZ,POP         ;IF NOT FOUND RETURN TO CALLER
0054      INC  HL             ;MOVE TO NEXT VALUE IN STRING
0055      LD   A,(HL)         ;GET NEXT VALUE AFTER CLS TOKEN
0056      SUB  30H            ;REDUCE IT TO RANGE 0-8
0057      JR   Z,EXEC         ;IF ZERO THEN EXECUTE COMMAND
0058      LD   B,8            ;LOAD B REG WITH UPPER LIMIT
0059 CMPR CP   B              ;CHECK IF A=B
0060      JR   Z,EXEC         ;IF YES THEN EXECUTE COMMAND
0061      DJNZ CMPR           ;REDUCE B AND CONTINUE CHECK
0062      JR   POP            ;NO MATCH SO RETURN TO CALLER
0063 EXEC POP  DE             ;RETRIEVE OLD STRING ADDRESS
0064      POP  DE             ;RETRIEVE OLD RETURN ADDRESS
0065      LD   DE,1D1EH       ;LOAD NEW RETURN ADDRESS
0066      PUSH DE             ;SAVE NEW RETURN ADDRESS
0067      INC  HL             ;MOVE TO NEXT VALUE IN STRING
0068      PUSH HL             ;SAVE CURRENT STRING ADDRESS
0069      ADD  A,A            ;MULTIPLY CLS
0070      ADD  A,A            ;VALUE BY 16 TO
0071      ADD  A,A            ;CALCULATE THE
0072      ADD  A,A            ;COLOUR OFFSET
0073      JR   NZ,SKIP        ;IF RESULT NOT ZERO THEN SKIP
0074      INC  A              ;IF ZERO INCREASE TO ONE
0075 SKIP ADD  A,7FH          ;ADD 127 TO GET GRAPHICS BLOCK
0076 ;
0077 ;CLEAR SCREEN ROUTINE
0078 ;
0079      LD   HL,7000H       ;LOAD START OF SCREEN ADDRESS
0080      LD   (7820H),HL     ;SET CURSOR POSITION
0081      LD   DE,7001H       ;LOAD START OF SCREEN PLUS ONE
0082      LD   BC,01FFH       ;NUMBER OF BYTES TO MOVE
0083      LD   (HL),A         ;LOAD GRAPHICS BLOCK INTO HL
0084      LDIR                ;DO A BLOCK FILL OF THE SCREEN
0085      POP  HL             ;RETRIEVE STRING ADDRESS
0086      RET                 ;RETURN TO 1D1EH TO CONTINUE
0087 ENDP DEFB 0              ;END OF PROGRAM MARKER
```

To have the command function properly, insert a colon between the THEN and the new command as below,

100 IF X=4 THEN:CLS4

Now, when X=4 the THEN part of the statement will be executed, including, as is usual, any additional commands in the remainder of the line. However, once the colon is reached, the BASIC ROM returns to its usual processing, via the RST 10H routine, and the CLS4 command is then interpreted on its own and not as part of the IF-THEN statement. This is the same solution suggested in the VZ-DOS manual, when using disk commands, which are affected in exactly the same way.

This is essentially the approach I have used to produce a

LISTING 3

```
100 REM ************************************************
110 REM # ENHANCED CLS COMMAND BY LARRY TAYLOR 1986    #
120 REM ************************************************
130 REM # CALCULATE THE NEW TOP OF MEMORY POINTER       #
140 REM ************************************************
150 NB=79:TM=(PEEK(30897)+PEEK(30898)*256)-NB
160 HB=INT(TM/256):LB=TM-HB*256
170 POKE30897,LB:POKE30898,HB
180 REM ************************************************
190 REM # RESET THE BASIC STACK POINTER                 #
200 REM ************************************************
210 CLEAR50
220 REM ************************************************
230 REM # LOCATION OF SET UP PROGRAM                    #
240 REM ************************************************
250 EB=31274
260 EH=INT((EB+1)/256):EL=EB+1-EH*256
270 REM ************************************************
280 REM # LOAD USER EXECUTION PROGRAM POINTER           #
290 REM ************************************************
300 POKE30862,EL:POKE30863,EH
310 REM ************************************************
320 REM # LOAD 23 BYTE SET UP PROGRAM                   #
330 REM ************************************************
340 FOR T=1TO23
350  READD
360  POKEEB+T,D
370  CS=CS+D
380 NEXT
390 REM ************************************************
400 REM # GET NEW TOP OF MEMORY AND MOVE TO NEXT LOCATION #
410 REM ************************************************
420 TM=PEEK(30897)+PEEK(30898)*256
430 IFTM>32767THENTM=TM-65536
440 REM ************************************************
450 REM # LOAD 79 BYTE ENHANCED CLS ROUTINE            #
460 REM ************************************************
470 FOR T=1TO79
480  READD
490  POKETM+T,D
500  CS=CS+D
510 NEXT
520 REM ************************************************
530 REM # IF DATA CHECKSUM VERIFIES EXECUTE SET UP PROGRAM #
540 REM ************************************************
550 IFCS<>10968THENPRINT"- ERROR IN DATA ENTRY -":END
560 X=USR(0)
570 REM ************************************************
580 REM # SET UP EXECUTION ROUTINE DATA IN DECIMAL FORM #
590 REM ************************************************
600 DATA 243,49,0,119,42,4,120,34,40,122,42,177,120,35,34,4,120
610 DATA 205,77,27,195,25,26
620 REM ************************************************
630 REM # ENHANCED CLS ROUTINE DATA IN DECIMAL FORM     #
640 REM ************************************************
650 DATA 217,33,91,29,209,183,237,82,213,217,194,120,29,229,205
660 DATA 120,29,32,7,225,237,91,40,122,213,201,254,132,32,245
670 DATA 35,126,214,48,40,9,6,8,184,40,4,16,251,24,230,209,209
680 DATA 17,30,29,213,35,229,135,135,135,135,32,1,60,198,127,33
690 DATA 0,112,34,32,120,17,1,112,1,255,1,119,237,176,225,201
```

VZ-EPSON Printer Patch, which enables all the normal printer functions for Epson or Epson-compatible printers. As well as providing the ability to LLIST and LPRINT all inverse and graphics characters, the COPY command is intercepted by the patch. As a result, its function has been enhanced to allow a proper dump of both the LO-RES and HI-RES screens. Corrections have been made to the flawed inverse character data, and when listing, the routine is capable of recognising all the hidden commands, which may have been entered using an Extended BASIC. The patch relocates to the top of available RAM and can be used with Steve Olney's EXTENDED BASIC, already resident in memory, enabling ready access to the functions of both. I hope that the techniques used here to produce what I have found to be an extremely useful utility will encourage others to attempt further such developments.

Perhaps additional enhancements to the VZ's BASIC could be explored. The Commodore 64 is served by a number of enhanced BASICs, why not the VZ? Programs which make use of such BASICs require that the language be loaded before they will function properly. However, this is little different to programs using disk commands needing the DOS to be interpreted correctly. Certainly, the opportunity exists to endow the humble VZ with a brand new bag of tricks.

For anyone interested, copies of the completed VZ-Epson Printer Patch, may be obtained on tape for $15, from:

J.C.E. D'Alton
VSOFTWAREZ
39 Agnes St
Toowong
Qld 4066

3 of 3